Algorithms 2020

NP-Hardness -
more reductions

Recap

- HW - oral grading Thurs. + Fri. - sign-up!
- Reading - over LPs.
- Canvas - some space issues (email if you need any files)
- Last HW: due before Thanksgiving

P, vs. NP, + co-NP

Consider only decision problems:

so Yes/No output

P Set of decision problems that can be solved in polynomial time.

?!! Ex essentially any thing we've seen (except backtracking, exp time problem)

NP Set of problems such that, if the answer is yes + you hand me ~~proof~~ certificate, I can verify/check in polynomial time. P

Ex: CIRCUIT SAT, sorting, searching, flow

Co-NP: Can verify a "No" answer.

CircuitSAT

Primality of a #:

no is easy!

$P \subseteq NP$

Dfn: NP-Hard

X is NP-Hard
$\Longleftrightarrow$
If X could be solved in polynomial time, then P=NP.

Recall P & NP?

So if <u>any</u> NP-Hard problem could be solved in polynomial time, then all of NP could be.

Initial ex:

CircuitSAT ←

To prove NP-Hardness of A:

Reduce a known NP-Hard problem to A.

convert it

call poly time

if poly time existed

NP-Hard problem

convert to instance of A (in poly time)

subroutine to solve problem A

Y

N.

Y

N

general instance of known problem

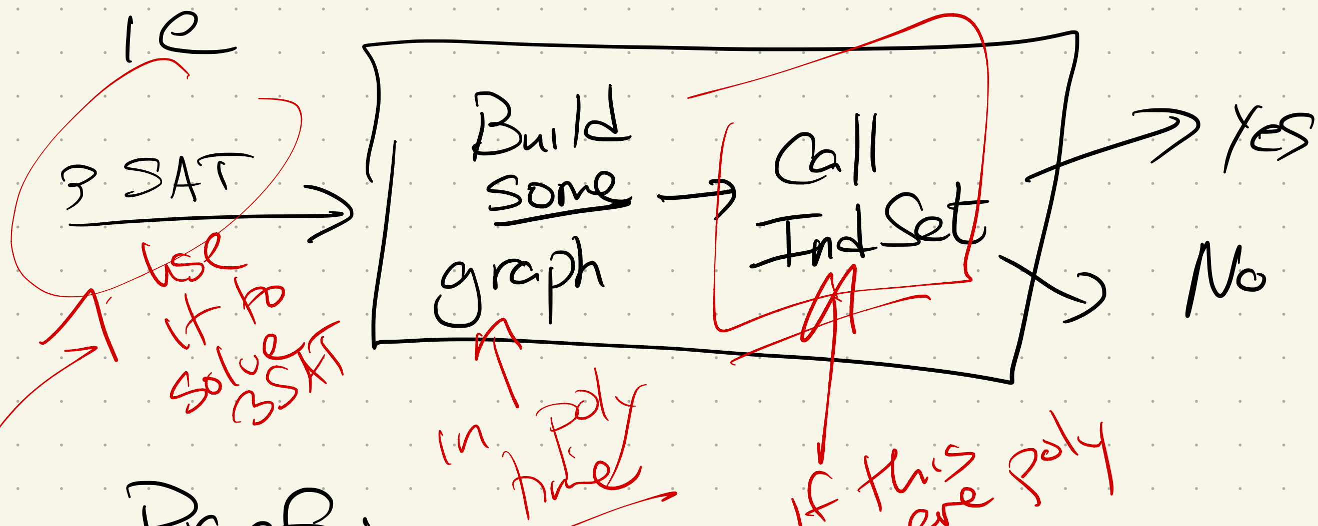
many of these

normally: to solve X, reduce to A & call subroutine

The Pattern: Reductions

1) Find an NP-Hard problem, & solve it using unknown problem as a subroutine

ie



Key: Proof:

Need if & only if!

(ie might be some wierd indep. set that doesn't make a SAT)

Challenge: Finding correct NP Hard problem

So far:

- CircuitSAT
- SAT
- 3SAT

logic-based; first studied examples

- Ind. Set
- Clique
- Vertex Cover

Graph problems

Today

- 3-Coloring
- Subset Sum (or Wed)

based problems

Next: Graph Coloring

{red, blue, green, pink, ...}

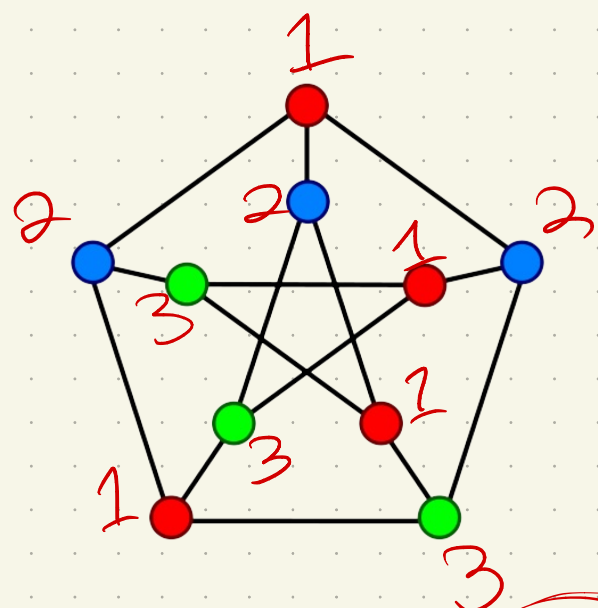
A k-coloring of a graph G is a map:

$$c : V \to \{1, \ldots, k\}$$

that assigns one of k "colors" to each vertex so that every edge has 2 different colors at its endpoints.

1, kth color

Goal: Use few colors



3-coloring

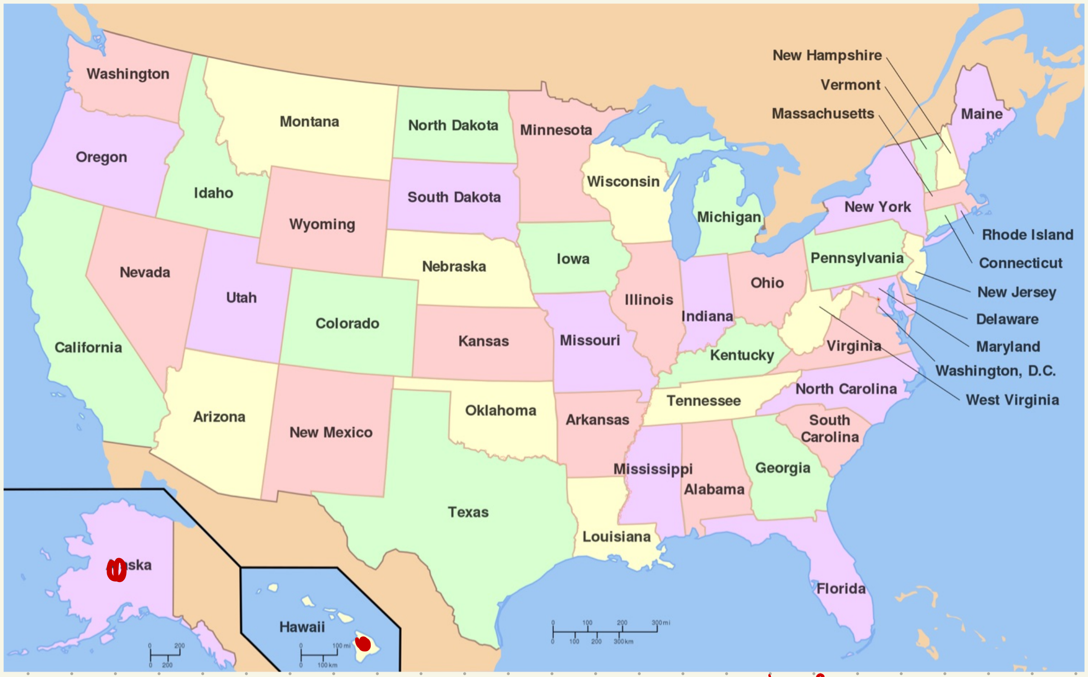
$k = 3$

Brute force: $\underbrace{k \cdot k \cdot \ldots \cdot k}_{n} = k^n$ possible colorings

exp-

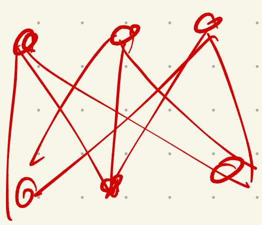
(backtracking)

Aside: this is famous!
Ever heard of map coloring?



? no 2 nbrs the same color

Famous theorem: every planar graph (ie map) is 4-colorable



not planar $K_{3,3}$

not planar

Thm: 3-colorability is NP-Complete.

(Decision version: Given G, output yes/no)

In NP:

certificate

If answer is yes, give me the coloring (a # per vertex from $\{1,2,3\}$)

Check each edge "uv" to see if $c(u) = c(v)$

$O(E)$

NP-Hard:

Reduction from 3SAT.

formula $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\cdots) \cdots$

Given formula for 3SAT Φ, we'll make a graph G_Φ.

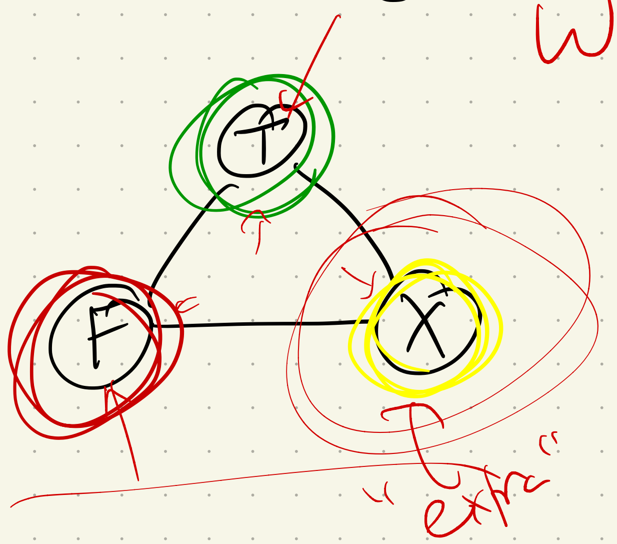
Φ = m clause, n variable

Φ will be satisfiable $\iff$ G_Φ can be 3-colored.

Key notion: Build "gadgets"!

① Truth gadget - one triangle

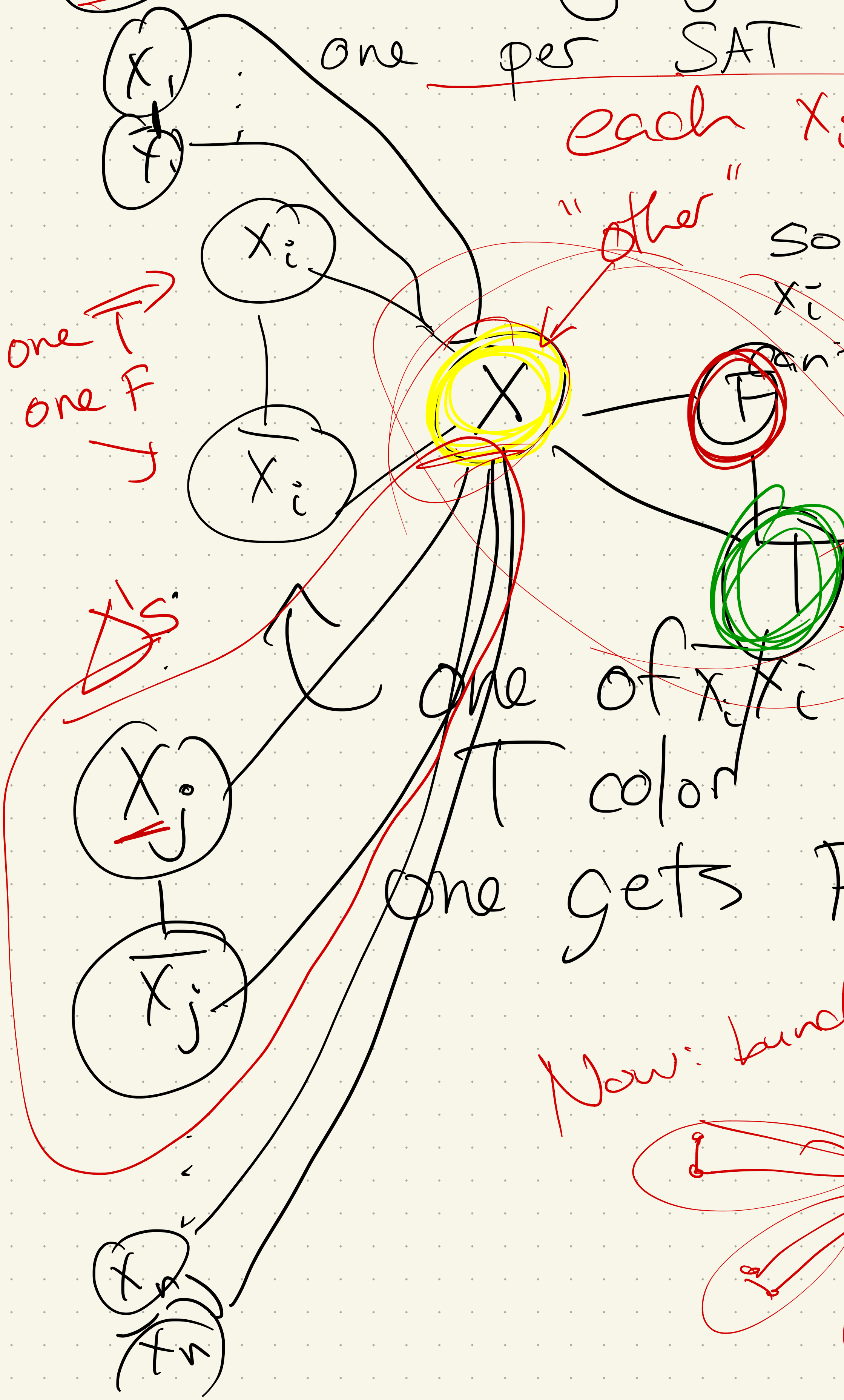
why?



Must use 3 colors - establishes a "true" color.

② Variable gadget — one per SAT variable

each x_i: x_i — $\bar{x}_i$



"other"

so both x_i + $\bar{x}_i$ can't be true

one T
one F

T/F △ gadget

△'s:

one of x_i/$\bar{x}_i$ gets T color
one gets F color

Now: bunch of △'s

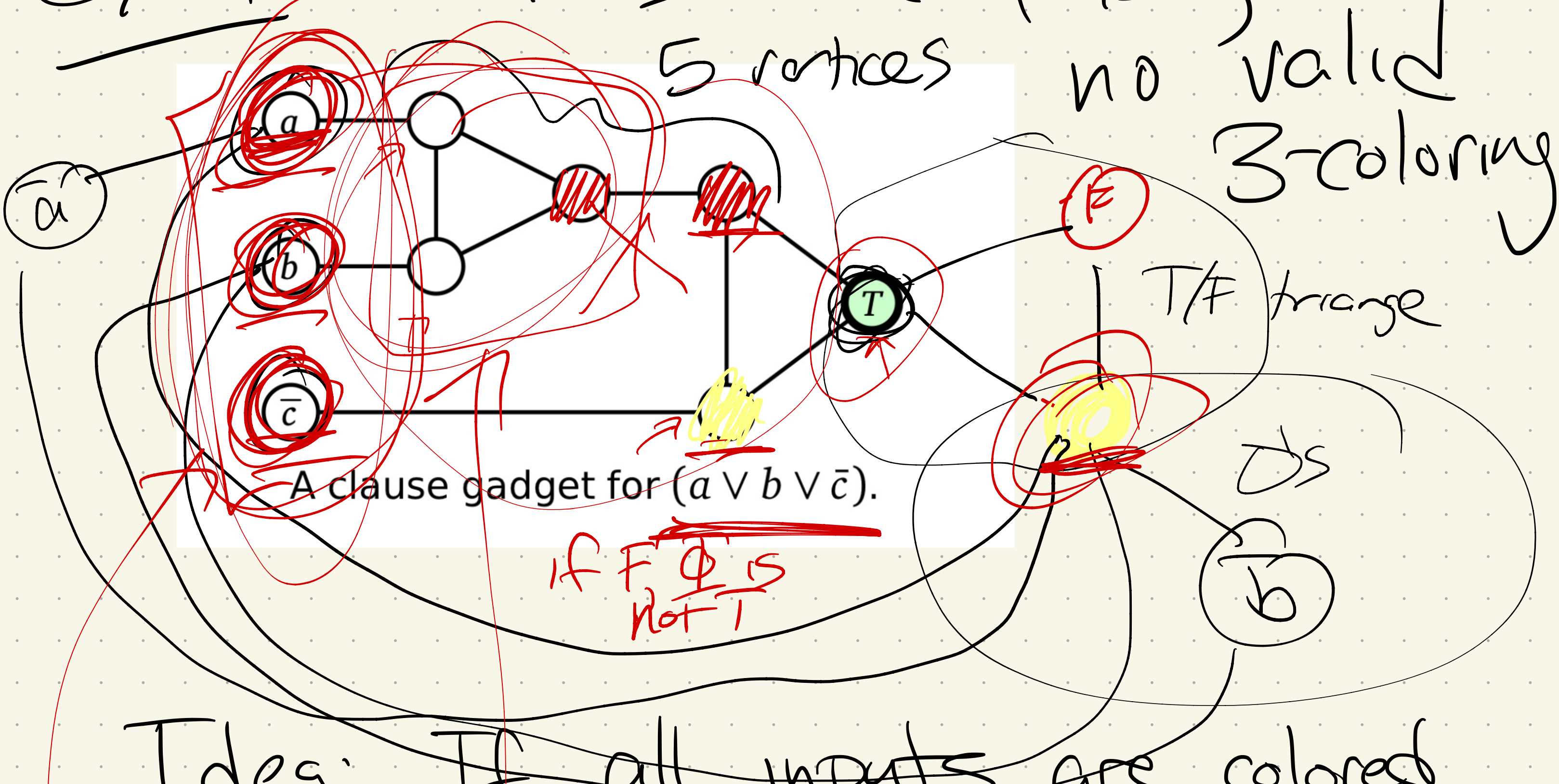
main T/F

(3) Clause gadget:

For each clause, join 3 of the variable vertices to the "true" vertex from the truth gadget.

Goal: If all 3 are false, no valid 3-coloring

5 vertices



A clause gadget for $(a \vee b \vee \bar{c})$.

if F Φ is not T

Idea: If all inputs are colored False, can't 3-color:

Any 3 coloring must make ~~one~~ of these green all connect to red, so impossible to 3-color

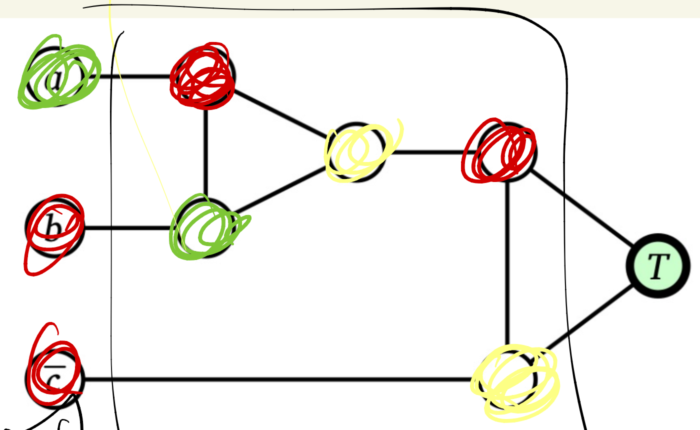
3 colorings of G_Φ $\Longleftrightarrow$ Φ is satisfiable

Pf:

$\Rightarrow$ 3 colorings:
Built G_Φ so that I can make each "green" be a true value, & each "red" be a false value.
No variable & its negation are same color, b/c connected by edge (& not "yellow" since in a $\triangle$ w/ yellow vertex)
If all in a clause are red, can't 3-color $\Rightarrow$ each clause has $\geq$ colored true

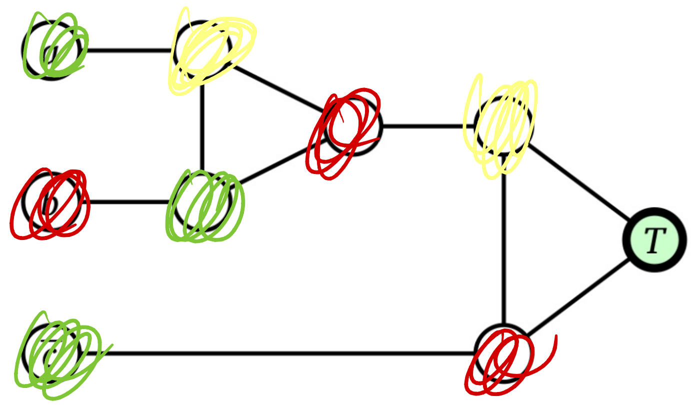
$\Leftarrow$: Φ satisfiable $\Rightarrow$ coloring (make each true x_i be green each false yellow, & extend coloring)

1



A clause gadget for $(a \vee b \vee \bar{c})$.

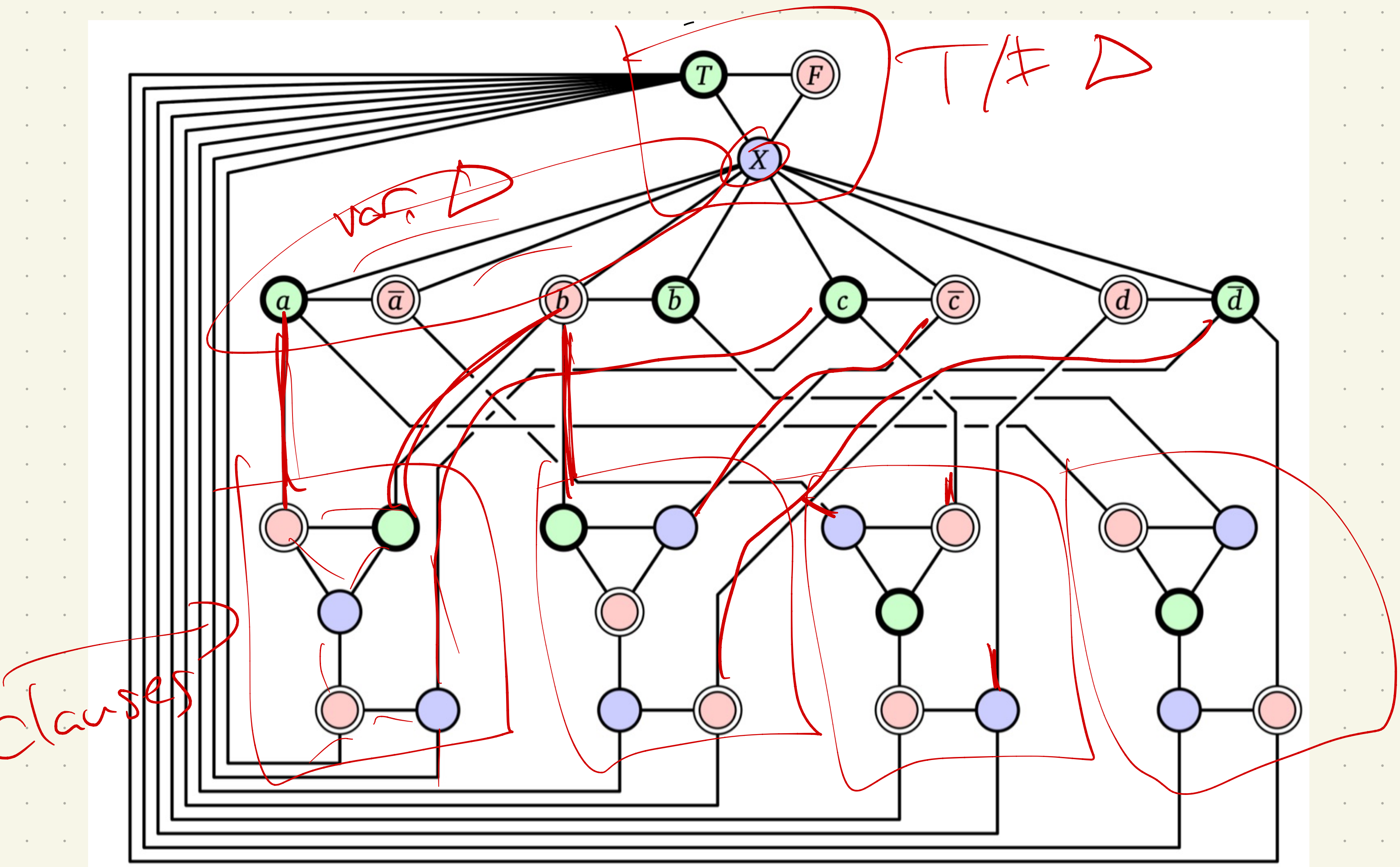
2



A clause gadget for $(a \vee b \vee \bar{c})$.

Cases (8 of them), but always color if not all red on x_i's

Final reduction image:



A 3-colorable graph derived from the satisfiable 3CNF formula
$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

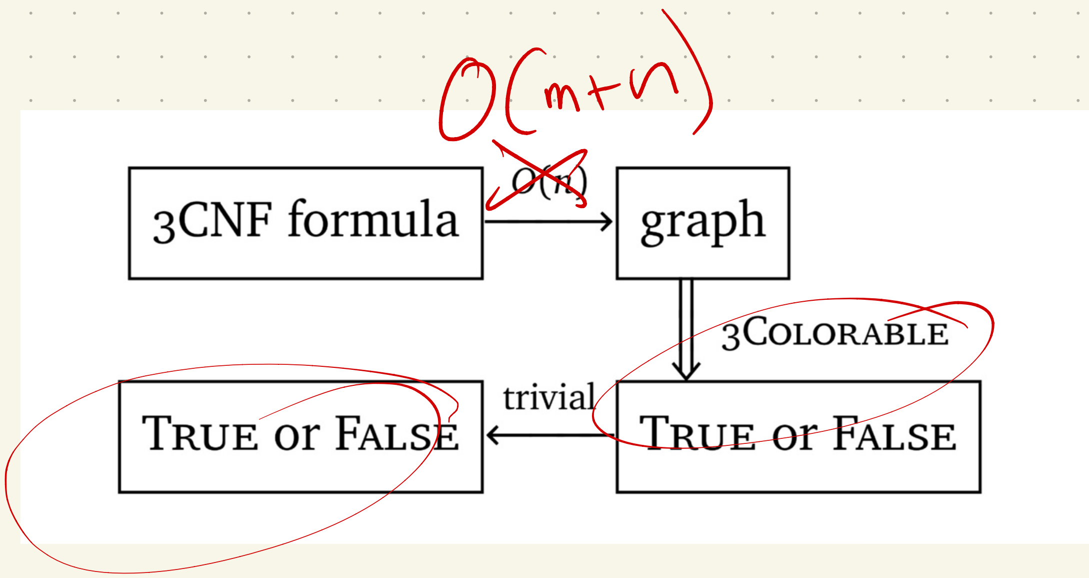
$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \ldots$

Time to build G_Φ:
(remember, need polynomial in formula size, $\underline{n} + m$)

$V = 3 + 2n + \underline{5m}$

$E = 3 + 3n + 11m$

So:



Subset Sum:

Given a set of numbers
$X = \{x_1, x_2, x_3, \ldots, x_n\}$
and a target t, does some subset of X sum to t?

log t bits in input

Ex: (actually did this one! See lecture from Ch. 2

Runtime:

backtracking: every # is either in set, or not

n #s

exponential

use DP: $O(nT)$

memoize!

size T

Subset Sum is NP-Hard.

Reduction: Vertex Cover

Input: Graph G & size k

Challenge: Need to construct a set of numbers, so that we hit some target sum if & only if a vertex cover in G of size k exists.

Recall: Base 4

$$31203 = 3\cdot 4^0 + 0\cdot 4^1 + 2\cdot 4^2 + 1\cdot 4^3 + 3\cdot 4^4$$

Idea: Use base 4:

force a target T that requires you to use only vertices, but to "cover" edges

Number edges $0 \ldots E-1$

↳ create a number for subset sum:

e_0: $b_0 = 1 = 4^0$

e_1: $b_1 = 4^1 = 4$

e_2: $b_2 = 4^2 = 16$

e_3: $b_3 = 4^3$

⋮

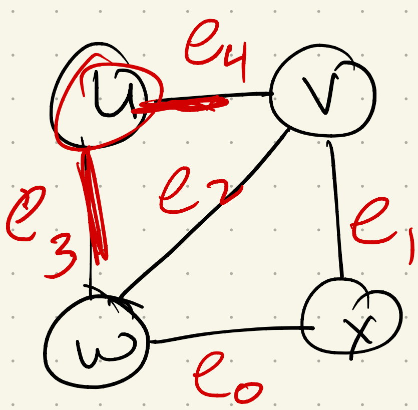
e_{E-1}: $b_{E-1} = 4^{E-1}$

#5

For each vertex, make another #:

$$a_v := \underbrace{4^E}_{> \text{ any edge } \# \, b_i} + \sum_{e_i \text{ into/out of } v} 4^i$$

Think of base 4 representation:



4^E

34

$a_u := 111000_4 = 1344$	$b_{uv} := 010000_4 = 256$
$a_v := 110110_4 = 1300$	$b_{uw} := 001000_4 = 64$
$a_w := 101101_4 = 1105$	$b_{vw} := 000100_4 = 16$
$a_x := 100011_4 = 1029$	$b_{vx} := 000010_4 = 4$
	$b_{wx} := 000001_4 = 1$

Now, set $T = k \cdot 4^{\bar{t}} + \sum_{i=0}^{\bar{t}-1} 2 \cdot 4^i$

<u>Why?</u>

<u>Proof</u>: size k VC $\Longrightarrow$ sum to T

<u>$\Longleftarrow$</u> VC : k vertices.

$\Leftarrow$: take the a_i's & b_i's that

$$= k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i$$

Time to reduce:

So: If I could solve Subset Sum in poly time, I could use it to solve VC.

$\Rightarrow$ Subset Sum is also NP-Hard
(& since in NP, also NP-Complete)

Next time:

More # ones,
+ a wrap-up.

Friday: On to LP!